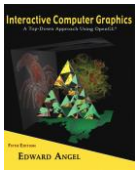


Name: _____ Term # ____

Homework 11 SOLUTIONS

(400 points)

NOTE: **Chapter 11 of the textbook shows the procedural methods of modeling that allows computer graphics to be used to model physical constraints and complex behavior of objects.**
Part A is intended to be done by hand.
Part B is an OpenGL application.

A. **(200 pts) Paper and Pencil**

(Guidelines: *Read the material from the textbook chapter, you can use textbook figures to exemplify your answer, use keywords, summarize your answer, but the answer* ***cannot be longer the 7 lines!****)*

11.1 ALGORITHMIC MODELS
a. Explain.
ANSWER:

11.2 PHYSICALLY-BASED MODELS AND PARTICLE SYSTEMS
a. Explain.
ANSWER:

11.3 NEWTONIAN PARTICLES
a. Explain.
ANSWER:

11.3.1 Independent PARTICLES
a. Explain.
ANSWER:

11.3.2 Spring Forces
a. Explain.
ANSWER

11.3.3 Attractive and Repulsive Forces
a. Explain.
ANSWER

11.4 SOLVING PARTICLE SYSTEMS
a. Explain.
ANSWER:

11.5 CONSTRAINTS
Explain
ANSWER:

11.5.1 Collisions
Explain
ANSWER:

11.5.2 Soft Constraints
Explain
ANSWER:

11.6 A SIMPLE PARTIAL SYSTEM
a. Explain.
ANSWER:

11.6.1 Displaying the Particles
a. Explain.
ANSWER:

11.6.2 Updating the Particle Positions
a. Explain.
ANSWER:

11.6.3 Initialization
a. Explain.
ANSWER:

11.6.4 Collisions
a. Explain.
ANSWER:

11.6.5 Forces
a. Explain.
ANSWER:

11.6.6 Flocking
a. Explain.
ANSWER:

11.7 LANGUAGE-BASED MODELS
a. Explain.
ANSWER:

11.8 RECURSIVE METHODS AND FRACTALS
a. Explain.
ANSWER:

11.8.1 Rulers and Length
a. Explain.
ANSWER

11.8.2 Fractal Dimensions
a. Explain.
ANSWER

11.8.2 Midpoint Division and Brownian Motion
a. Explain.
ANSWER

11.8.4 Fractal Mountains
a. Explain.
ANSWER

11.8.5 The Mandelbrot Set
a. Explain.
ANSWER

11.9 PROCEDURAL NOISE
a. Explain.
ANSWER:

B. (200 pts) Visual Studio 2008 C++ Project

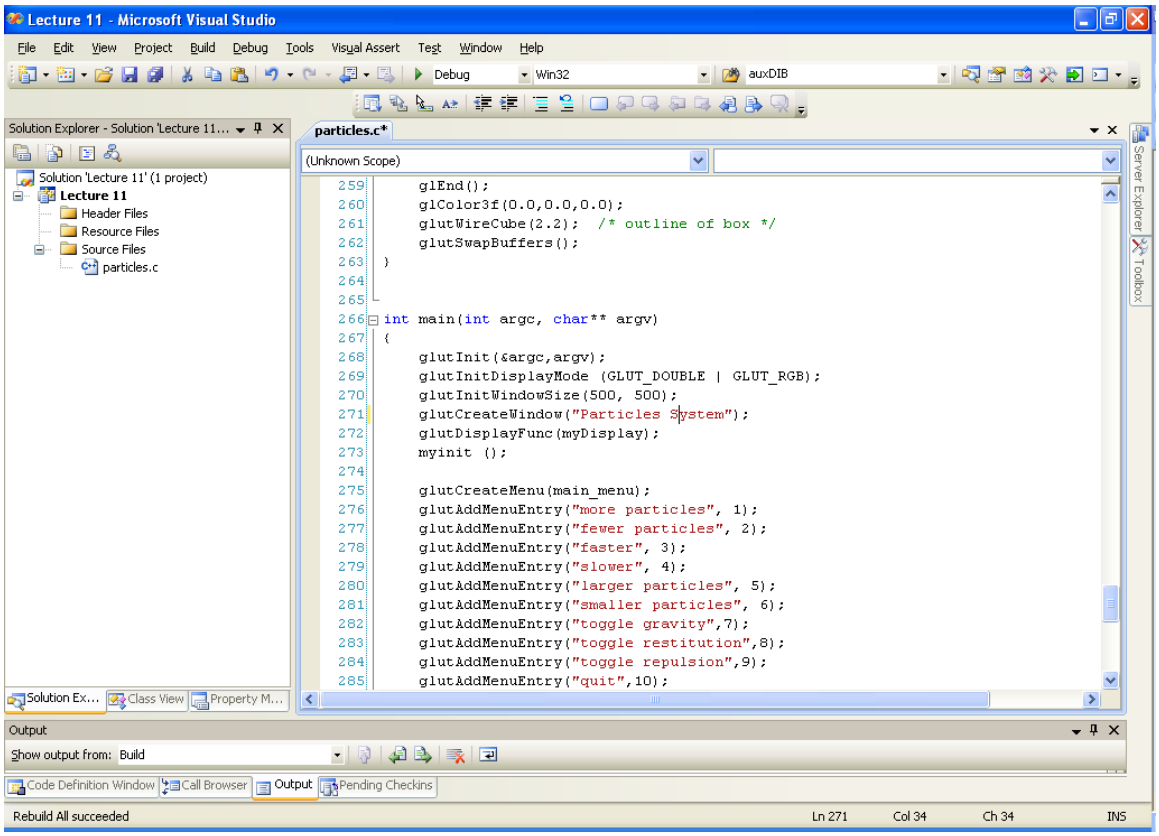
B1. Create Visual Studio 2008 C++, Empty Project, Homework11:

Modify the particles.c from Class Participation on Lecture 11:

1. `glutCreateWindow("HOMEWORK 11 Particle System");`
2. `Instead of particles in a "box" modify the program for particles in a "sphere".`
3. `Add code that when the particles colide with each other an explosion or fireworks effect will take place.`

Build and run this Project: Insert a screenshot of your output.

ANSWER:




```
// particles.c

/* Particles in a box */

#define MAX_NUM_PARTICLES 1000
#define INITIAL_NUM_PARTICLES 25
#define INITIAL_POINT_SIZE 5.0
#define INITIAL_SPEED 1.0
```

```
typedef int bool;
#define TRUE 1
#define FALSE 0

#include <stdlib.h>
#include <stdio.h>

#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

void myDisplay();
void myIdle();
void myReshape(int, int);
void main_menu(int);
void collision(int);
float forces(int, int);

void myinit();

/* globals */

int num_particles; /* number of particles */

/* particle struct */

typedef struct particle
{
     int color;
     float position[3];
     float velocity[3];
     float mass;


} particle;

particle particles[MAX_NUM_PARTICLES]; /* particle system */

/* initial state of particle system */

int present_time;
int last_time;
int num_particles = INITIAL_NUM_PARTICLES;
float point_size = INITIAL_POINT_SIZE;
float speed = INITIAL_SPEED;
bool gravity = FALSE; /* gravity off */
bool elastic = FALSE; /* restitution off */
bool repulsion = FALSE; /* repulsion off */
float coef = 1.0; /* perfectly elastic collisions */
float d2[MAX_NUM_PARTICLES][MAX_NUM_PARTICLES]; /* array for
interparticle distances */

GLsizei wh = 500, ww = 500; /* initial window size */

GLfloat colors[8][3]={{0.0, 0.0, 0.0}, {1.0, 0.0, 0.0},{0.0, 1.0, 0.0},
```

```
    {0.0, 0.0, 1.0}, {0.0, 1.0, 1.0}, {1.0, 0.0, 1.0}, {1.0, 1.0, 0.0},
    {1.0, 1.0, 1.0}};

/* rehaping routine called whenever window is resized or moved */

void myReshape(int w, int h)
{

/* adjust clipping box */

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-2.0, 2.0, -2.0, 2.0, -4.0, 4.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(1.5,1.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0);

/* adjust viewport and  clear */

        if(w<h) glViewport(0,0,w,w);
        else glViewport(0,0,h,h);


/* set global size for use by drawing routine */

        ww = w;
        wh = h;
}

void myinit()
{
        int  i, j;

 /* set up particles with random locations and velocities */

        for(i=0; i<num_particles; i++)
        {
            particles[i].mass = 1.0;
            particles[i].color = i%8;
            for(j=0; j<3; j++)
            {
                particles[i].position[j] = 2.0*((float)
rand()/RAND_MAX)-1.0;
                particles[i].velocity[j] = speed*2.0*((float)
rand()/RAND_MAX)-1.0;
            }
        }
        glPointSize(point_size);


/* set clear color to grey */

        glClearColor(0.5, 0.5, 0.5, 1.0);
}

void myIdle()
{
```

```
    int i, j, k;
    float dt;
    present_time = glutGet(GLUT_ELAPSED_TIME);
    dt = 0.001*(present_time -  last_time);
    for(i=0; i<num_particles; i++)
    {
       for(j=0; j<3; j++)
       {
           particles[i].position[j]+=dt*particles[i].velocity[j];
           particles[i].velocity[j]+=dt*forces(i,j)/particles[i].mass;
       }
        collision(i);
    }
    if(repulsion) for(i=0;i<num_particles;i++) for(k=0;k<i;k++)
    {
            d2[i][k] = 0.0;
            for(j=0;j<3;j++) d2[i][k]+= (particles[i].position[j]-
              particles[k].position[j])*(particles[i].position[j]-
              particles[k].position[j]);
            d2[k][i]=d2[i][k];
    }
    last_time = present_time;
    glutPostRedisplay();
}

float forces(int i, int j)
{
   int k;
   float force = 0.0;
   if(gravity&&j==1) force = -1.0; /* simple gravity */
   if(repulsion) for(k=0; k<num_particles; k++)  /* repulsive force */
   {
      if(k!=i) force+= 0.001*(particles[i].position[j]-
particles[k].position[j])/(0.001+d2[i][k]);
   }
   return(force);
}

void collision(int n)

/* tests for collisions against cube and reflect particles if necessary
*/

{
     int i;
     for (i=0; i<3; i++)
     {
           if(particles[n].position[i]>=1.0)
           {
                particles[n].velocity[i] = -
coef*particles[n].velocity[i];
                particles[n].position[i] = 1.0-
coef*(particles[n].position[i]-1.0);
           }
           if(particles[n].position[i]<=-1.0)
           {
```

```
                particles[n].velocity[i] = -
coef*particles[n].velocity[i];
                particles[n].position[i] = -1.0-
coef*(particles[n].position[i]+1.0);
            }
      }
}

void main_menu(int index)
{
   switch(index)
   {
      case(1):
      {
                num_particles = 2*num_particles;
                myinit();
                break;
      }
      case(2):
      {
                num_particles = num_particles/2;
                myinit();
           break;
      }
      case(3):
      {
                speed = 2.0*speed;
                myinit();
           break;
      }
        case(4):
        {
                speed = speed/2.0;
                myinit();
                break;
        }
        case(5):
        {
                point_size = 2.0*point_size;
                myinit();
                break;
        }
        case(6):
        {
                point_size = point_size/2.0;
                if(point_size<1.0) point_size = 1.0;
                myinit();
                break;
        }
        case(7):
        {
                gravity = !gravity;
                myinit();
                break;
        }
        case(8):
        {
```

```
                elastic = !elastic;
                if(elastic) coef = 0.9;
                  else coef = 1.0;
                myinit();
                break;
        }
        case(9):
        {
                repulsion = !repulsion;
                myinit();
                break;
        }
      case(10):
      {
            exit(0);
            break;
      }
   }
}

void myDisplay()
{
    int i;

    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POINTS); /* render all particles */
    for(i=0; i<num_particles; i++)
    {
       glColor3fv(colors[particles[i].color]);
       glVertex3fv(particles[i].position);
    }
    glEnd();
    glColor3f(0.0,0.0,0.0);
    glutWireCube(2.2);  /* outline of box */
    glutSwapBuffers();
}


int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Particles System");
    glutDisplayFunc(myDisplay);
    myinit ();

    glutCreateMenu(main_menu);
    glutAddMenuEntry("more particles", 1);
    glutAddMenuEntry("fewer particles", 2);
    glutAddMenuEntry("faster", 3);
    glutAddMenuEntry("slower", 4);
    glutAddMenuEntry("larger particles", 5);
    glutAddMenuEntry("smaller particles", 6);
    glutAddMenuEntry("toggle gravity",7);
    glutAddMenuEntry("toggle restitution",8);
    glutAddMenuEntry("toggle repulsion",9);
```

```
    glutAddMenuEntry("quit",10);
    glutAttachMenu(GLUT_MIDDLE_BUTTON);

    glutIdleFunc(myIdle);
    glutReshapeFunc (myReshape);
    glutMainLoop();
}
```